Assignment 3

BIOSTAT 213: Introduction to Computing in the R Software Environment

Due: Wednesday, March 6, 2019

Collaboration with other students in this course is permitted on this assignment. However, you must write all work—including R code—in your own voice, as much as possible, and acknowledge your collaborators as thoroughly as possible.

The assignment is due online, in the Collaborative Learning Environment, by 9 am, Wednesday, March 6, 2019. You *must* submit a Word-compatible document or a PDF file.

You will be using the `nhanes.csv` and `unps.csv` data files for this assignment. The questions below do *not* assume any of the modifications to the data from lab or slides. In other words, you may begin this assignment by simply re-importing the CSV files.

For each of the below, share your code as well as the output (if there is any output). If your answer to a question is unclear from your R output, or difficult to identify, include an answer in text form (in addition to the code and output).

1. *Use* ***`unps.csv`****.* The UNPS data are in long format: each individual can appear on more than one row. One way to convert the data to wide format is to do so "manually." This problem will walk you through such a process. In short, we will create a data frame for each of the 3 interview waves (`wave`), and then merge the 3 data sets together by individual (`pid`).

   (a) Create and store a subset of the data, keeping only individuals interviewed (`wave`) in 2009. Repeat this process for individuals interviewed in 2010 and individuals interviewed in 2011. You should have 3 data sets. Use `nrow()` to show the number of rows in each data frame. You should get the following numbers:

   ```
   > nrow(uu1)
   [1] 1928
   > nrow(uu2)
   [1] 1566
   > nrow(uu3)
   [1] 1518
   ```

   (b) Merge the 2009 and 2010 data frames, treating the 2009 data frame as the `x` data frame. Use object assignment to store this merged data frame. You should use the following arguments:

   - `by`: the name of the identification variable for individuals, in quotes (`'pid'`)
   - `suffixes=c('','2010')`: suffixes to add to the variable names, so that we can differentiate the waves (the result is that the `x` data frame gets no suffix and the `y` data frame gets the `.2010` suffix)
   - `all=TRUE`: necessary because some individuals entered the study after 2009 (note for example that if we instead use `all.x=TRUE, all.y=FALSE`, we would only be keeping the individuals from 2009 and excluding the individuals who entered the study in 2010)

   Now, repeat this process: merge this merged data frame and the 2011 data frame by individual (`pid`), treating the merged data frame from above as the `x` data frame. Modify the `suffixes` argument to be `c('','.2011')`.

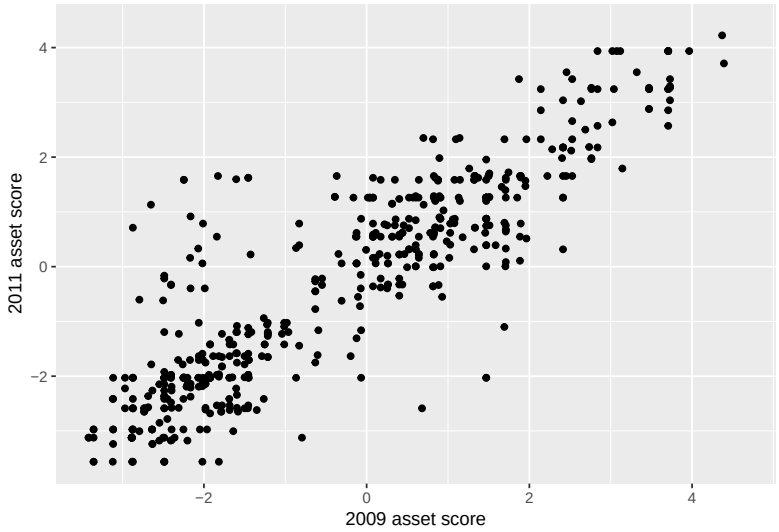
   Use `dim()` to examine the dimensions of your merged data frame. Also use `head()`, `tail()`, and `names()` to check the variable names.

   ```
   > dim(ww.manual)
   [1] 3222   64
   ```

```
> head(names(ww.manual))
[1] "pid"         "hhid"        "age_months" "day"         "month"
[6] "year"
> tail(names(ww.manual))
[1] "numberofchildren.2011" "numberofmen.2011"      "numberofwomen.2011"
[4] "selfrepdrought.2011"   "urb.2011"              "male.2011"
```

If you've added columns to the UNPS data, you might have a different number of columns than shown above.

(c) Use your merged data frame and ggplot2 to create a scatter plot of 2009 asset score (should be `assetscore`) and 2011 asset score (should be `assetscore.2011`), with the 2009 asset score on the x axis. Your plot should look like this:



2. *Use `unps.csv`.* In the above, we reshaped the UNPS data from long to wide "manually." An alternative is to use `reshape()`. You need only 4 arguments:

   - `data`: the original UNPS data frame
   - `timevar`: the name of the variable, in quotes, indicating time (you can use `'wave'` or `'year'`)
   - `idvar`: the name of the identification variable for individuals, in quotes (`'pid'`)
   - `direction`: the direction of reshaping, in quotes (in this case, `'wide'`)

   Reshape the original UNPS data to wide format using `reshape()`.

   Use `dim()` to examine the dimensions of your merged data frame. Also use `head()`, `tail()`, and `names()` to check the variable names:

```
> dim(ww)
[1] 3222   61
> head(names(ww))
[1] "pid"             "hhid.2009"       "age_months.2009" "day.2009"
[5] "month.2009"      "year.2009"
> tail(names(ww))
[1] "numberofchildren.2011" "numberofmen.2011"      "numberofwomen.2011"
[4] "selfrepdrought.2011"   "urb.2011"              "male.2011"
```

   If you've added columns to the UNPS data, you might have a different number of columns than shown above. You should, however, see the same number of rows above, and it should match the number of rows from the wide data frame you created "manually." The number of columns below should be

3 fewer than the number of columns in the data frame you created "manually" because `reshape()` automatically drops the time variable (it is no longer necessary because the values are added as suffixes to the variable names).

3. *Use* ***unps.csv***. Use the original UNPS data and the year (`year`), month (`month`), and day (`day`) variables to create a single variable for interview date. Your variable should be a Date object. Use `head()` the first few values of your date variable and `class()` to show the object type. Hint: you might begin by using `paste()` to paste the year, month, and day values together (not necessarily in that order), as in Assignment 1. For example, you may end up with character strings that look like: `2009/12/15`.

4. *Use* ***presidencies.csv***. Use `tapply()` to find the total duration, in days, that each party (`party`) has held office.

5. *Use* ***nhanes.csv***. The `apply()` function can be very useful for iteratively modifying data. For example, we've noticed that the measurements of diastolic blood pressure in the NHANES data include values equal to 0:

```
> table(ii$bpxdi1==0,useNA='always')

FALSE  TRUE  <NA>
 5083    28   658
> table(ii$bpxdi2==0,useNA='always')

FALSE  TRUE  <NA>
 5251    47   471
> table(ii$bpxdi3==0,useNA='always')

FALSE  TRUE  <NA>
 5249    58   462
> table(ii$bpxdi4==0,useNA='always')

FALSE  TRUE  <NA>
  391     3  5375
```

We've agreed that these values may be errors, and that we may treat them as missing values. From slide 31, we can changes values less or equal to 10 mmHg to missing as follows:

```
> is.na(ii$bpxdi1)<-(ii$bpxdi1<=10)
```

We could do this for each of the 4 measurements, which is not too tedious, but could be fairly tedious if we had even more variables to recode. To use `apply()` to iteratively recode the variables:

```
make.missing<-function(vv,cutoff,direction='le'){
  if(direction=='le'){
    is.na(vv)<-(vv<=cutoff)
  }
  if(direction=='ge'){
    is.na(vv)<-(vv>=cutoff)
  }
  vv
}
ii[,c('bpxdi1','bpxdi2','bpxdi3','bpxdi4')]<-apply(
  ii[,c('bpxdi1','bpxdi2','bpxdi3','bpxdi4')],2,make.missing,cutoff=10)
```

In `make.missing()`, we included a `direction` argument to include an option for making missing values greater or equal to some cutoff missing. This function is thus generalizable; imagine, for example, variables in which values greater or equal to 7 represent "don't know" or "skipped." The default direction

is less or equal to. We included `vv` in the last line because we want the function to return the modified variable.

In `apply()`, the `2` is short for `MARGIN=2`. We've also added the `cutoff` argument from `make.missing()`.

To the left of the object assignment, we've included the original variables. The code will thus write over the original variables. If you want to create new variables instead (preserving the original variables), you can use different variable names on that left-hand side. For example: `ii[,c('di1','di2','di3','di4')]`.

Copy the code block above (I've excluded prompts and plus signs for your convenience), paste it into your script, and run it.

Now, check each of the variables for values less than or equal to 10.

6. Write a function that implements the quadratic formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The symbol $\pm$ means plus or minus. For example, 2 ± 3 means $2 - 3$ and $2 + 3$.

Your function should accept 3 arguments (a, b, and c) and return a vector containing 2 numbers. Note that, in general, functions will return whatever is on the last line. So, if `x` is on the last line, the function will return `x`.

Include the following checks:

```
qf(2,-11,14)
[1] 2.0 3.5
qf(1,2,-63)
[1] -9  7
qf(1,-16,48)
[1]  4 12
```

7. *Use* ***nhanes.csv.*** Use `tapply()` to find the median weight (`bmxwt`) of each age group (`agegroup`), but do this only for the males (`gender`). One way to build in the stratification by males is to use `with()`. For example, a re-written version of the example on slide 58 of Module 6, adding `with()`, is:

```
with(data=ii,tapply(bmxbmi,bpcat,mean,na.rm=TRUE))
          Elevated Hypertensive crisis              Normal
          29.47767           26.13673           27.67927
           Stage 1            Stage 2
          30.41979           29.88089
```

As with numerous problems in Assignment 2, we can build in stratification by modifying the `data` argument.

8. *Use* ***nhanes.csv.*** Use `apply()` to find the median weight (`bmxwt`) and height (`bmxht`) of all females (`gender`). Note that `with()` is not necessary here: the first argument to `apply()`, `X`, is the data.